Module submission header

Submission preparation instructions

Completion of this header is mandatory, subject to a 2-point deduction to the assignment. Only add plain text in the designated areas, i.e., replacing the relevant 'NA's. You must fill out all group member Names and Drexel email addresses in the below markdown list, under header **Module submission group**. It is required to fill out descriptive notes pertaining to any tutoring support received in the completion of this submission under the **Additional submission comments** section at the bottom of the header. If no tutoring support was received, leave NA in place. You may as well list other optional comments pertaining to the submission at bottom. *Any distruption of this header's formatting will make your group liable to the 2-point deduction.*

Module submission group

- Group member 1
  - Name: Kasonde Chewe
  - Email: kc3745@drexel.edu
- Group member 2
  - Name: Ghanath V
  - Email: gv347@drexel.edu
- Group member 3
  - Name: Kholoud Hamed M Al Nazzawi
  - Email: ka974@drexel.edu
- Group member 4
  - Name: NA
  - Email: NA

Additional submission comments

- Tutoring support received: NA
- Other (other): NA

Assignment Group 1

Problem C *(50 points)*

This problem deals with finding "pangrams" in text. A pangram is a sentence containing all 26 letters of the alphabet. `x` and `y` in the cell below are example sentences, `x` is a pangram, `y` is not.

In [1]:
```
x = "Jim quickly realized that the beautiful gowns are expensive."
```

```
y = "This sentence is most certainly not a pangram."
```

C1. *(5 points)* Complete the function, `indices()`, that takes a string as input and outputs a list of the index numbers where each lowercase character from the English alphabet occurs first (only) in the string. Note: your function should `break` and `return` the list if `26` characters are found.

[**Hint:** you must keep track of which characters have been `found`, and a `set` could help with this. Also, you can compare letters like numbers. For example, `char >= "a"` is a valid conditional statement. You can use this to check whether characters in a string are letters of the alphabet.]

In [2]:
```
# C1:Function(5/5)

def indices(text):
    first_indices = []; text = text.lower()

    #---your code starts here---
    import string

    all_letters = list(string.ascii_lowercase)
    #print(all_letters)
    counter = 0;
    new_list = []
    #index_list = []
    for i,v in enumerate(text):

        if(counter == 26):
            break
        if ((v in all_letters) and (v not in new_list)):
            first_indices.append(i)
            counter += 1
            new_list.append(v)

    #---your code stops here---
    return first_indices
```

For reference, your output should be:

```
[0, 1, 2, 4, 5, 7, 8, 9, 10, 12, 13, 14, 17, 19, 21, 22, 30, 36,
40, 41, 42, 43, 44, 51, 52, 57]
```

In [3]:
```
# C1:SanityCheck

first_indices = indices(x)
print(first_indices)
```

```
[0, 1, 2, 4, 5, 7, 8, 9, 10, 12, 13, 14, 17, 19, 21, 22, 30, 36, 40, 41, 42, 4
3, 44, 51, 52, 57]
```

C2. *(3 points)* Now complete the function `verify()`, which must take a string as input and use the output of the `indices()` function to check if the string is a pangram, where the output of `verify()` should should be boolean `True` or `False` named `has_all`.

In [4]:
```python
def verify(text):
    has_all = False
    first_indices = indices(text)

    #---your code starts here---
    import string


    has_all = True if len(first_indices) == 26 else False

    #---your code stops here---

    return has_all
```

For reference, your output should be:

```
    (True, False)
```

In [5]:
```python
# C2:SanityCheck

verify(x), verify(y)
```

Out[5]:
```
(True, False)
```

C3: *(2 points)* Now complete the below function named `tiny_verify()` that performs the check in a single line of code—*without using* `indices()`. [**Hint:** Use a comprehension.]

In [6]:
```python
# C3:Function(2/2)

def tiny_verify(text):

    #---your single line of code starts here---

    #---your code starts here---
    has_all =  all([True if letter in text.lower() else False for letter in 'abc

    #---your single line of code stops here---

    return has_all
```

For reference, your output should be:

```
    (True, False)
```

In [7]:
```python
# C2:SanityCheck

tiny_verify(x), tiny_verify(y)
```

Out[7]:
```
(True, False)
```

C4. *(5 points)* Now complete the `verify_missing()` by filling the `letters` set with any alphabetic characters that appear in the purported pangram. This version will return a list of missing letters instead of a boolean value, with the list of missing characters computed as a set difference against the `all_letters` set. [**Hint:** Use the string containing all the letters in the alphabet, imported from the `string` module.]

In [8]:
```
# C4:Function(5/5)
from string import ascii_lowercase as ascii_letters

def verify_missing(text):
    all_letters = set(ascii_letters)
    letters = set()

    # iterate over the input text and add each alphabetic character to the `lett
    for char in text.lower():
        if char in ascii_letters:
            letters.add(char)

    # compute the set difference between `all_letters` and `letters` to get the
    missing_letters = list(all_letters - letters)

    # sort the missing_letters list
    missing_letters.sort()

    return missing_letters
```

For reference, your output should be:

```
([], ['z', 'v', 'k', 'd', 'u', 'j', 'w', 'b', 'f', 'x', 'q'])
```

In [9]:
```
verify_missing(x), verify_missing(y)
```

Out[9]:
```
([], ['b', 'd', 'f', 'j', 'k', 'q', 'u', 'v', 'w', 'x', 'z'])
```

C5. *(5 points)* Now iterate through the loaded list of pangrams in `data/pangrams.txt` and `verify` which are actually pangrams. Store any incomplete sentences (that *don't* `verify`) in the `imposters` list, which forms the function's output.

In [10]:
```
# C5:Function(5/5)

potential_pangrams = open("data/pangrams.txt", "r", encoding='UTF-8').readlines(
potential_pangrams = [sentence.strip().lower() for sentence in potential_pangram

def evaluate_pangrams(sentences):
    imposters = []

    for sentence in sentences:
        if not verify(sentence):
            imposters.append(sentence)

    return imposters
```

For reference, your output should be:

```
Faulty pangram:
Show mangled quartz flip vibe exactly.
Missing letters:
['k', 'j']
```

```
Faulty pangram:
Unamazingly, this six-word pangram is questionable!
Missing letters:
['c', 'v', 'k', 'j', 'f']
```

In [11]:

```
# C5:SanityCheck

imposters = evaluate_pangrams(potential_pangrams)
for sentence in imposters:
    print("Faulty pangram: ")
    print(sentence)
    print("Missing letters: ")
    print(verify_missing(sentence))
    print()
```

```
Faulty pangram:
show mangled quartz flip vibe exactly.
Missing letters:
['j', 'k']

Faulty pangram:
unamazingly, this six-word pangram is questionable!
Missing letters:
['c', 'f', 'j', 'k', 'v']
```

C6: *(3 points)* Now complete the function below to use the output from the `verify()` function to fix the failed pangrams by any means necessary, and then collect the original and fixed sentences in a list of tuples.

In [12]:

```
# C6:Function(3/3)

# C6:Function(3/3)

import re
import random as ra
from collections import defaultdict, Counter
import re
from string import ascii_lowercase as ascii_letters

def tokenize(text, space = True, wordchars = "a-zA-Z0-9-'"):
    tokens = []
    for token in re.split("(["+wordchars+"]+)", text):
        if not space:
            token = re.sub("[ ]+", "", token)
        if not token:
            continue
        if re.search("["+wordchars+"]", token):
            tokens.append(token)
        else:
            tokens.extend(token)
    return tokens

def fix_pangrams(sentences, seed = 511):
    word_counts = Counter([t for sentence in sentences for t in tokenize(sentenc
    word_index = defaultdict(list)
    for w in word_counts: word_index[w[0]].append(w)
    imposters_fixed = []
    ra.seed(seed)
    #---your code starts here---
    all_letters = list(ascii_letters)

    for row in potential_pangrams:
        result = verify(row)
        if not result:
            temp = row

            row = re.sub(r".$", ";",row)
            temp2 = ""
            for i in all_letters:
                if i not in temp:
                    temp2 = i + temp2
                    temp3 = row + temp2
                    tup = (temp,temp3)
            imposters_fixed.append(tup)

    #---your code stops here---

    return imposters_fixed
```

For reference, your output could be:

```
    Faulty pangram:
    show mangled quartz flip vibe exactly.
    Fixed pangram:
    show mangled quartz flip vibe exactly; kvetching jets.

    Faulty pangram:
    unamazingly, this six-word pangram is questionable!
    Fixed pangram:
    unamazingly, this six-word pangram is questionable; chimp veldt
    kazakh jonquils flip!
```

In [13]:
```
# C6:SanityCheck
imposters_fixed = fix_pangrams(potential_pangrams)
for sentence, fixed in imposters_fixed:
    print("Faulty pangram: ")
    print(sentence)
    print("Fixed pangram:")
    print(fixed)
    print()
```

```
Faulty pangram:
show mangled quartz flip vibe exactly.
Fixed pangram:
show mangled quartz flip vibe exactly;kj

Faulty pangram:
unamazingly, this six-word pangram is questionable!
Fixed pangram:
unamazingly, this six-word pangram is questionable;vkjfc
```

C7. *(5 points)* In the cell below, complete the metadata munging job using the information about the set of books in the `books_available` string. Assuming there may be many more books and that their information can be provided in the same format, create a data object that holds the book authors and titles associated to each book number, and writes the metadata as a JSON file in the `data/books/` directory using the following schema:

`books = { BookNumber: { 'author': AuthorName 'title': BookTitle, ... }, ... }`

In [14]:
```
# C7:Inline(5/5)

import json
import re

books_available = """
84.txt; Frankenstein, or the Modern Prometheus; Mary Wollstonecraft (Godwin) She
98.txt; A Tale of Two Cities; Charles Dickens
161.txt; Sense and Sensibility; Jane Austen
730.txt; Oliver Twist or the Parish Boy's Progress; Charles Dickens
768.txt; Wuthering Heights; Emily Brontë
1322.txt; Leaves of Grass; Walt Whitman
1342.txt; Pride and Prejudice; Jane Austen
1400.txt; Great Expectations; Charles Dickens
2701.txt; Moby Dick; or the Whale; Herman Melville
4300.txt; Ulysses; James Joyce
"""

books = {}
metadata_file = "data/books/metadata.json"

#---your code starts here---
pattern = r"(\d+)\.txt;\s*(.*?);\s*(.*)"

for match in re.finditer(pattern, books_available):
    book_number = match.group(1)
    book_title = match.group(2)
    book_author = match.group(3)
    books[book_number] = {'author': book_author, 'title': book_title}
#---your code stops here---

json.dump({k: books[k] for k in sorted(books.keys())},
          open(metadata_file, "w"))

books
```

Out[14]:
```
{'84': {'author': 'Mary Wollstonecraft (Godwin) Shelley',
  'title': 'Frankenstein, or the Modern Prometheus'},
 '98': {'author': 'Charles Dickens', 'title': 'A Tale of Two Cities'},
 '161': {'author': 'Jane Austen', 'title': 'Sense and Sensibility'},
 '730': {'author': 'Charles Dickens',
  'title': "Oliver Twist or the Parish Boy's Progress"},
 '768': {'author': 'Emily Brontë', 'title': 'Wuthering Heights'},
 '1322': {'author': 'Walt Whitman', 'title': 'Leaves of Grass'},
 '1342': {'author': 'Jane Austen', 'title': 'Pride and Prejudice'},
 '1400': {'author': 'Charles Dickens', 'title': 'Great Expectations'},
 '2701': {'author': 'or the Whale; Herman Melville', 'title': 'Moby Dick'},
 '4300': {'author': 'James Joyce', 'title': 'Ulysses'}}
```

For reference, you *could* literally start typing:

```
books = {84 : {'title': 'Frankenstein, or the Modern
Prometheus',
               'author': 'Mary Wollstonecraft (Godwin)
Shelley'}, ...}
```

but that would miss the point. Instead, you should use regular expressions to process the `books_available` object automatically by any convenient delimiters to produce the target object structure.

C8. *(4 points)* Now complete the `sentokenize(text)` function using the `re` (regular expressions) module to split the book text into sentences using the `re.split(pattern, string)` function. The function must take a document and break it into 'sentences' (sentence-like strings), and then break these into lists of tokens within the larger `sentences` list.

Note: you can efficiently use the below pattern to obtain the desired output for sentokenization:

- `"(\s*(?<=["+delims+"][^"+sentchars+"])\s*)"`

Additionally, ensure each `sentence` stored from the output of the sentokenization pattern is non-empty and processed by `tokenize` before being placed into the `sentences` list.

In [15]:
```
# C8:Function(4/4)
def sentokenize(text, space=True, delims=".?!", sentchars="a-zA-Z0-9-',;:"):
    sentences = []

    #---your code starts here---
    pattern = r"(\s*(?<=[" + delims + "][^" + sentchars + "])\s*)"
    raw_sentences = re.split(pattern, text)

    for raw_sentence in raw_sentences:
        if raw_sentence.strip():
            sentence = tokenize(raw_sentence, space=space)
            if space:
                sentences.append(sentence)
            else:
                sentences.extend(sentence)
    #---your code stops here---

    return sentences
```

For reference, your output should be:

```
['Only', ' ', 'add', ' ', 'plain', ' ', 'text', ' ', 'in', ' ',
'the', ' ', 'designated', ' ', 'areas', ',', ' ', 'i', '.', 'e',
'.', ',', ' ', 'replacing', ' ', 'the', ' ', 'relevant', ' ',
"'NA's", '.', ' ', '\n']
['\nCompletion of this header is mandatory, subject to a 2-point
deduction to the assignment. \n',
 "Only add plain text in the designated areas, i.e., replacing
the relevant 'NA's. \n",
 'You must fill out all group member Names and Drexel email
addresses in the below markdown list, \nunder header Module
submission group. ',
 'It is required to fill out descriptive notes pertaining to
\nany tutoring support received in the completion of this
submission \nunder the Additional submission comments section at
the bottom of the header. \n',
 'If no tutoring support was received, leave NA in place. \n',
 'You may as well list other optional comments pertaining to the
submission at bottom. \n',
 "Any distruption of this header's formatting will make your
group liable to the 2-point deduction.\n"]
```

In [16]:

```
# C8:SanityCheck

test_sentences = sentokenize("""
Completion of this header is mandatory, subject to a 2-point deduction to the as
Only add plain text in the designated areas, i.e., replacing the relevant 'NA's.
You must fill out all group member Names and Drexel email addresses in the below
under header Module submission group. It is required to fill out descriptive not
any tutoring support received in the completion of this submission
under the Additional submission comments section at the bottom of the header.
If no tutoring support was received, leave NA in place.
You may as well list other optional comments pertaining to the submission at bot
Any distruption of this header's formatting will make your group liable to the 2
""")

print(test_sentences[1])
["".join(sentence) for sentence in test_sentences]
```

```
['Only', ' ', 'add', ' ', 'plain', ' ', 'text', ' ', 'in', ' ', 'the', ' ', 'de
signated', ' ', 'areas', ',', ' ', 'i', '.', 'e', '.', ',', ' ', 'replacing', '
', 'the', ' ', 'relevant', ' ', "'NA's", '.']
```

Out[16]:

```
['\nCompletion of this header is mandatory, subject to a 2-point deduction to t
he assignment.',
 "Only add plain text in the designated areas, i.e., replacing the relevant 'N
A's.",
 'You must fill out all group member Names and Drexel email addresses in the be
low markdown list, \nunder header Module submission group.',
 'It is required to fill out descriptive notes pertaining to \nany tutoring sup
port received in the completion of this submission \nunder the Additional submi
ssion comments section at the bottom of the header.',
 'If no tutoring support was received, leave NA in place.',
 'You may as well list other optional comments pertaining to the submission at
bottom.',
 "Any distruption of this header's formatting will make your group liable to th
e 2-point deduction."]
```

C9. *(3 points)* Now use the `sentokenize` function to complete the `get_pangrams(book_num)` function to determine which sentences in a given book are pangrams. The output of this function should be a tuple, continaing the list of found `pangrams` and the total number of sentencs (`num_sentences`) in the book. [Hint: don't forget to `.lower()` your sentences before attempting pangram verification!]

In [17]:

```
# C9:Function(3/3)
def is_pangram(sentence):
    alphabet = set("abcdefghijklmnopqrstuvwxyz")
    return alphabet.issubset(set(sentence.lower()))
```

```python
def get_pangrams(book_num):
    pangrams = []
    num_sentences = 0
    with open("data/books/" + str(book_num) + ".txt", "r") as f:
        for sentence in sentokenize(f.read()):

            #---your code starts here---
            num_sentences += 1
            sentence_str = ' '.join(sentence).lower()
            if is_pangram(sentence_str):
                pangrams.append(sentence)
            #---your code stops here---

    return pangrams, num_sentences
```

For reference, your output should be:

```
8363 1 Every town-gate and village taxing-house had its band of
citizen-patriots, with their national muskets in a most
explosive state
of readiness, who stopped all comers and goers, cross-questioned
them,
inspected their papers, looked for their names in lists of their
own,
turned them back, or sent them on, or stopped them and laid them
in
hold, as their capricious judgment or fancy deemed best for the
dawning
Republic One and Indivisible, of Liberty, Equality, Fraternity,
or
Death.
```

In [18]:
```python
# C9:SanityCheck
book_results = get_pangrams(98)
print(book_results[1], len(book_results[0]), "".join(book_results[0][0]) if book
```

```
8364 1 Every town-gate and village taxing-house had its band of
citizen-patriots, with their national muskets in a most explosive state
of readiness, who stopped all comers and goers, cross-questioned them,
inspected their papers, looked for their names in lists of their own,
turned them back, or sent them on, or stopped them and laid them in
hold, as their capricious judgment or fancy deemed best for the dawning
Republic One and Indivisible, of Liberty, Equality, Fraternity, or
Death.
```

C10. *(8 points)* Now complete the below function to determine who is the pangrammiest author and what the pangrammiest book is, as determined by most pangrams per sentence. For this part, you must use the `defaultdict` s to store data in the `pangrams_by_author` and `pangrams_by_book` objects. The first object (`pangrams_by_author`) should be keyed by `author` , and the second object (`pangrams_by_book`) should be keyed by `(title, book_num)` -tuples. Each object's values should be total list of pangrams and sentence numbers for each grouping of the data.

In [19]:
```
# C10:Function(4/8)


def collect_pangrams(books):
    pangrams_by_author = defaultdict(lambda: [[], 0])
    pangrams_by_book = defaultdict(lambda: [[], 0])

    #---your code starts here---
    for book_num, book_info in books.items():
        author = book_info['author']
        title = book_info['title']

        book_pangrams, num_sentences = get_pangrams(book_num)

        pangrams_by_author[author][0].extend(book_pangrams)
        pangrams_by_author[author][1] += num_sentences

        pangrams_by_book[(title, book_num)][0].extend(book_pangrams)
        pangrams_by_book[(title, book_num)][1] += num_sentences

    #---your code stops here---
    return pangrams_by_author, pangrams_by_book
```

For reference, your output should be:

```
(2, 26836)
```

In [20]:
```
# C10:SanityCheck

pangrams_by_author, pangrams_by_book = collect_pangrams(books)

len(pangrams_by_author["Charles Dickens"][0]), pangrams_by_author["Charles Dicke
```

Out[20]:
```
(2, 26842)
```

Now complete the function below to operate on the `pangrams_by_author` and `pangrams_by_book` objects and compute the portion of all sentences that were pangrams for each author-grouping and book.

In [22]:
```
# C10:Function(4/8)
# make corrections

def compute_pangram_rates(pangrams_by_author, pangrams_by_book):
    authors_pangrams_per_sentence, books_pangrams_per_sentence = [], []

    for author, value in pangrams_by_author.items():
        pangrams, num_sentences = value
        pangram_rate = len(pangrams) / num_sentences
        authors_pangrams_per_sentence.append((pangram_rate, author))

    # Loop through each
    for book, value in pangrams_by_book.items():
        pangrams, num_sentences = value
        pangram_rate = len(pangrams) / num_sentences
        books_pangrams_per_sentence.append((pangram_rate, book[0]))

    return authors_pangrams_per_sentence, books_pangrams_per_sentence
```

For reference, your output should be:

```
In order of decreasing pangrammyness, the authors are:
[(0.00234375, 'Walt Whitman'),
 (0.0004151617054842861, 'James Joyce'),
 (0.0003994407829039345, 'Herman Melville'),
 (0.00034928396786587494, 'Emily Brontë'),
 (0.00023796303640834457, 'Jane Austen'),
 (7.452675510508272e-05, 'Charles Dickens'),
 (0.0, 'Mary Wollstonecraft (Godwin) Shelley')]

In order of decreasing pangrammyness, the books are:
[(0.00234375, 'Leaves of Grass'),
 (0.0004233700254022015, 'Pride and Prejudice'),
 (0.0004151617054842861, 'Ulysses'),
 (0.0003994407829039345, 'Moby Dick; or the Whale'),
 (0.00034928396786587494, 'Wuthering Heights'),
 (0.0001268874508311128, "Oliver Twist or the Parish Boy's
Progress"),
 (0.00011957431543704412, 'A Tale of Two Cities'),
 (0.0, 'Great Expectations'),
 (0.0, 'Sense and Sensibility'),
 (0.0, 'Frankenstein, or the Modern Prometheus')]
```

In [23]:
```
# C10:SanityCheck

from pprint import pprint

authors_pangrams_per_sentence, books_pangrams_per_sentence = compute_pangram_rat

print("In order of decreasing pangrammyness, the authors are: " )
pprint(sorted(authors_pangrams_per_sentence,reverse = True))

print("\nIn order of decreasing pangrammyness, the books are: " )
pprint(sorted(books_pangrams_per_sentence, reverse=True))
```

```
In order of decreasing pangrammyness, the authors are:
[(0.00234375, 'Walt Whitman'),
 (0.0004067189978443893, 'James Joyce'),
 (0.0003982873643333665, 'or the Whale; Herman Melville'),
 (0.00034928396786587494, 'Emily Brontë'),
 (0.00023784983746927773, 'Jane Austen'),
 (7.4510096118024e-05, 'Charles Dickens'),
 (0.0, 'Mary Wollstonecraft (Godwin) Shelley')]

In order of decreasing pangrammyness, the books are:
[(0.00234375, 'Leaves of Grass'),
 (0.0004233700254022015, 'Pride and Prejudice'),
 (0.0004067189978443893, 'Ulysses'),
 (0.0003982873643333665, 'Moby Dick'),
 (0.00034928396786587494, 'Wuthering Heights'),
 (0.0001268874508311128, "Oliver Twist or the Parish Boy's Progress"),
 (0.00011956001912960307, 'A Tale of Two Cities'),
 (0.0, 'Sense and Sensibility'),
 (0.0, 'Great Expectations'),
 (0.0, 'Frankenstein, or the Modern Prometheus')]
```

C11. *(7 points)* Finally, complete the below function to compute the most efficient pangram and its author and book, as determined by fewest characters per pangram.

In [24]:
```python
# C11:Function(7/7)

def most_efficient_pangram(pangrams_by_book):
    most_efficient_author = None
    most_efficient_book = None
    most_efficient_title = None
    least_characters = float("Inf")
    best_pangram = "NA"

    #---your code starts here---
    for (author, book_num), pangram_info in pangrams_by_book.items():
        pangrams, _ = pangram_info
        for pangram in pangrams:
            num_characters = len(pangram)
            if num_characters < least_characters:
                least_characters = num_characters
                best_pangram = pangram
                most_efficient_author = author
                most_efficient_book = book_num
                most_efficient_title = books[book_num]['title']
    #---your code stops here---

    return (most_efficient_book, most_efficient_author,
            most_efficient_title, least_characters, best_pangram)
```

For reference, your output should be:

```
The best pangramming author, Charles Dickens, wrote a 223-
character pangram in the book:
"Oliver Twist or the Parish Boy's Progress" (booknumber: 730)

This pangram was:
At least half a
dozen more were severally drawn forth from the same box, and
```

```
surveyed
with equal pleasure; besides rings, brooches, bracelets, and
other
articles of jewellery, of such magnificent materials, and costly
workmanship, that Oliver had no idea, even of their names.
```

In [25]:

```
# C11:SanityCheck

(most_efficient_book, most_efficient_author,
 most_efficient_title, least_characters, best_pangram) = most_efficient_pangram(

print("The best pangramming author, " + most_efficient_author +
      ", wrote a " + str(least_characters) +
      "-character pangram in the book: \n\"" + most_efficient_title +
      "\" (booknumber: " + str(most_efficient_book) +")\n\n" +
      "This pangram was: \n" + "".join(best_pangram))
```

```
The best pangramming author, Oliver Twist or the Parish Boy's Progress, wrote a
97-character pangram in the book:
"Oliver Twist or the Parish Boy's Progress" (booknumber: 730)

This pangram was:
At least half a
dozen more were severally drawn forth from the same box, and surveyed
with equal pleasure; besides rings, brooches, bracelets, and other
articles of jewellery, of such magnificent materials, and costly
workmanship, that Oliver had no idea, even of their names.
```